# 1. Introduction

## 1.1. Project Description

The goal of this project is to provide Natural Language Processing (NLP) services in the form of APIs. NLP is one of the major tasks in Machine Learning (ML). Whenever someone finds themselves wanting a computer program that can carry out NLP tasks for them, they would have to go through very time consuming and difficult processes of data collection, data cleaning, finding people skilled enough to do something with that data, build an effective model that can ultimately be used to do the NLP tasks. Instead, an individual or an organization can make use of the APIs provided as a result of this project do these tasks. All they have to do is make API calls with their inputs and the API returns a JSON object with predictions. This is going to save the users months of time and effort. Users can make calls using any language that supports fetching JSON data from the browser. This gives them incredible flexibility while building their applications. It also allows them to focus more of their manpower on their main products. There is always a rising need for machines more powerful, intelligent, and faster than us to take over the tedious and repetitive tasks that we just don’t want to do. ML has provided solutions exactly to these tasks over the years. If someone is conducting a survey on a certain topic, they would not want to spend hours looking at tweets and determining the general sentiment about that topic. They would rather feed that information to a program and let it tell them what kind of sentiment the tweets are showing. The tricky part is getting that program to work properly and accurately. The project uses three different models, trained for three different tasks (Sentiment Analysis, Category Prediction, Spam Detection). All of them averaging an accuracy of over 94%. So, accuracy isn’t a problem anymore. The project was built on node.js using a tensorflow.js integration. This means most of the tasks are done on the browser itself, making it faster and requiring very less computational power.

### 1.1.1. Problem Definition

Text classification is a very important task in supervised machine learning. A piece of text is assigned to one or more classes or categories. This can be done manually or with the help of powerful machine learning algorithms. The problem with doing this manually is that it takes up a lot of time and resources. Let’s say you own a blogging website or a news website. Every article that is being posted has to be classified and put into a category. Making people read these articles manually is both time consuming and expensive. It would be easier if the computer itself classified these articles, as soon as they are posted. This is where the need for Natural Language Processing arises. Natural Language Processing or NLP, is a Machine Learning (ML) task that is used to train an ML model to recognize text data and get meaningful insights from it. This means that a trained ML model will be able to go through some text data and give us some context on it. So, if you pass an article as input, this model will be able to tell you where it belongs. NLP can also be used to do other interesting tasks such as Sentiment Analysis. This means that a model will be able to tell if some text data is positive, negative, or neutral about any topic that is in discussion. Our phones and email accounts are bombarded with spam every day. The only way to filter out the spam is by either making users flag the messages as spam manually or filter out the messages at the server end itself using an effective program. Context Analyzer provides solutions to all three of these tasks.

### 1.1.2. Purpose

The problem of classifying text can be done by the organizations or the users themselves. To do that they would have to start collecting data, hire skilled ML engineers, spend a lot of money and time building out an effective and accurate model. Then some more time has to be spent tuning the model to perform better on all kinds of data. All of this can be avoided when companies just use the APIs provided by this project. The purpose of this project was to automate cartain NLP tasks and also provide them as services through APIs. Integrating APIs is much cheaper and faster than building an entire team to carry out these tasks.

### 1.1.3. Scope

The main users of this project will be those looking to integrate NLP tasks into their application with very little human effort or resources and get very efficient results. The web application also provides a UI which can be used to carry out these tasks for one time users. The sentiment analysis API can be used for predicting sentiment of data from social media platforms, reviews on products, etc.. Category prediction is a multi-class classification task that can be used on news articles or blogs to classify them into different classes like politics, entertainment, sports, health, etc.. The spam detection API allows users to determine whether a message (any form of message; like SMS, email etc.) is spam. Initially I identified these three tasks as the major ones. The application is built in a way that it is always easy to add new features or build new APIs and add them.

### 1.1.4. Proposed Solution

The application consists of three different models all built using a Convolutional Neural Network, or CNN. The three models are trained on; a news dataset, the IMDB reviews dataset, and an SMS Spam dataset. The news dataset is going to be used to train the model for the multi class classification task. The IMDB reviews dataset is going to be used to train the model for sentiment analysis. The SMS spam dataset is used to train the model for spam detection. These models, once put into production will be able to do these classification tasks in mere seconds. This will also be cheaper and more effective.

To provide APIs, Node.js will be used. This has very good integration for tensorflow.js. Which means all ML tasks can be done on the browser itself. Node.js is also an amazing javascript runtime that can be used to build highly efficient endpoints. The web application can also be used as an external tool for the classification tasks. Once the dataset is loaded, the preprocessing starts.

First I use the stop words function to remove the prepositions like “the, of, he, she etc”. It also simplifies the words, for example if there are multiple words like “doing, did, done” it will be converted to “do”. Then the tokenizer is used to turn the text into sequence of numbers. The neural network takes only numbers as input. Once the preprocessing is done, the model is created with several layers. The output layer returns a value denoting which class the text belongs to. Once the model is trained and saved, tensorflow js is used to load it. After this, everything is done on the browser. The predictions are made at the node endpoints.

# 2. Literature Survey

## 2.1. Background Study

A lot of research goes into NLP almost every day. Providing APIs for NLP tasks is a bit of a complex task. One such way to build NLP APIs is using a robust back-end technology like Node.js and making use of its brilliant integration with Tensorflow.js. Tensorflow.js allows us to interact with ML models directly from the browser. This makes it currently one of the best libraries for building NLP APIs. Tensorflow.js, for a while was running only on experimental Node.js. Recently it was ported to latest stable build of Node.js. It shows that this is a good way to move forward.

### 2.1.1. Existing Systems

Google Cloud Natural Language provides NLP APIs. They also make use of tensorflow.js to provide the service. OpenNLP and Stanford NLP provide NLP libraries that can be integrated with supported languages. TextRazor provides NLP APIs but they do not use tensorflow.js. All these systems have certain drawbacks and gaps that can be filled with Context Analyzer API. The drawbacks and the solutions for them are mentioned in a future section.

### 2.1.2. Related Work

**Title**: Recurrent Neural Network for Text Classification with Multi-Task Learning

**Authors**: Pengfei Liu, Xipeng Qiu, Xuanjing Huang

**Summary**: Neural network based methods have obtained great progress on a variety of natural language processing tasks. However, in most previous works, the models are learned based on single-task supervised objectives, which often suffer from insufficient training data. In this paper, we use the multitask learning framework to jointly learn across multiple related tasks. Based on recurrent neural network, we propose three different mechanisms of sharing information to model text with task-specific and shared layers. The entire network is trained jointly on all these tasks. Experiments on four benchmark text classification tasks show that our proposed models can improve the performance of a task with the help of other related tasks.

#### **Title**: Long Short-Term Memory

#### **Authors**: Sepp Hochreiter, Jurgen Schmidhuber

**Summary**: Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is O(1). Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

### 2.1.3. Drawbacks of Existing Systems

Unlike Google Cloud Natural Language, this project allows users to customize the API results however they want. The response from the API is a JSON object, which means users can fetch any data they want from the output easily. Also, making API calls is easier than ever as all it requires is the input to be appended to the URL. (for ex: “localhost:3000/api/sentiment?predict=your+input”). A simple GET request would give you the predictions in a JSON response. While Open NLP and Stanford NLP provide good libraries, it wont be easy for every user to download the library and write code to make use of it. It is much easier to just make API calls and get output almost instantly. The major problem that Context Analyzer is solving is the ease of use and integration. As stated above, making API calls is very easy and simple for everyone to understand. The JSON response is also very easy to decode and every part of the response can be fetched individually based on the user’s requirements.

## 2.2. Feasibility Study

### 2.2.1. Technical Feasibility

The application runs completely on the browser. This does not require any additional installations or downloads. It is extremely easy to add new APIs or features when built. The back-end uses all the es6 features of Node.js which means it is highly responsive and fast. Obviously, the development part is hidden completely from the user. All the user will see is a responsive UI.

### 2.2.2. Economic Feasibility

The development of this web application barely costs anything. It can be built with computers that have a decent amount of computational power. I decided to use the CPU version of tensorflow and tensorflow.js. This means it can be built using devices without a GPU. All the tools and technologies used were open-source, so, no licensing was required.

### 2.2.3. Operational Feasibility

Operating the web application is also very easy. The web application is built completely on the browser, so it will be fast and responsive. The application also contains documentation for the APIs. Every new user can go through it and find out how to use the APIs. One time users can also make use of the UI provided for each of the APIs. This takes in the user input and returns predictions. This is effective to show as a demo for the APIs and gives the users an idea of how quickly the results are given.

**2.3. Tools and Technologies**

**2.3.1. Tensorflow / tfjs-node**

**Tensorflow** is an end-to-end open-source platform for ML. It has a wide range of tools and libraries, along with community resources that help us build state-of-the-art ML powered applications. In the context of this project, tensorflow was used to build and train the models. **Tensorflow.js** is an ML library for JavaScript. It allows us to develop ML models in JavaScript and use ML directly in the browser or in **Node.js.** This application made the most use of tensorflow and tfjs-node as it was the core component in building the APIs and models.

**2.3.2. Node.js**

**Node.js** is an asynchronous event-driven JavaScript runtime. **Node.js** allows us to build scalable network applications. **Node.js** served as the main back-end for this project for its **REST API** and **asynchronous** capabilities. A lot of asynchronous code is written to support the loading of models and making predictions. **Node.js** powers all of these tasks effectively.

**2.3.3. Python 3**

**Python** is a powerful and fast programming language. Python is user-friendly and easy to learn. Python basically runs on any platform and is open-source. Python was largely used as the platform for building and training the models. Tensorflow integrated with Python is the most powerful tool for ML. Python’s endless libraries facilitate users to perform almost any task.

**2.3.4. Postman**

**Postman** is an API development tool that helps users test and build powerful APIs. From headers to authentication tokens to simple JSON values, Postman supports all these features. It was largely used in the development part of the web application. All the routes and requests were simulated, monitored, and verified using Postman. Only then would they be integrated with the front-end.

**2.3.5. Handlebars.js / hbs**

**Handlebars.js** is used to build semantic templates. Handlebars is compatible with **Moustache** templates as well. Handlebars templates are compiled into JavaScript functions. These templates allow for reusability and dynamic rendering of content. **Hbs** is an express.js view engine for **Handlebars.js.** Hbs can be used with express.js to seamlessly write handlebars templates and integrate them with the HTML code. Hbs supports Partials and Views. Views are your main HTML pages and Partials support rendering of partial content on the views. This makes for building an effective, dynamic, and lightweight front-end.

# 3. Hardware and Software Requirements

## 3.1. Hardware Requirements

* **Development Environment:**
  + **HDD:** 4GB
  + **OS:** Ubuntu 18.04 LTS, Windows 10, Mac OSx
  + **Processor:** Intel i5 Gen 8
  + **RAM:** 8GB

## 3.2. Software Requirements

* **Server Side:** Node.js 13.x, Express 4.x, tfjs-node 1.5.x
* **Client-Side:** Handlebars 4.x, HTML5, CSS3
* **Environment:** Ubuntu 18.04, Windows 10, Node, Python runtime environment
* **Code Editor / Ide:** VS Code, Jupyter Lab

# 4. Software Requirement Specification

A summary of the program in the development process. It sets out functional and non-functional specifications. It is the basic collaborative document between the customer and the developer, listing necessary and specifications for the development of the project.

## 4.1. Users

Users are generally the people who visit the website, logs in, and then uses it for building machine learning models.

**The administrator** is any user who visits the website and interacts with the application. They can either use the application for building models or building APIs for production

**Customer** is the end-user who used the API that is built by the administrator to build their applications

## 4.2. Functional Requirements

In software engineering and systems engineering, the functional requirement specifies the function of the device or its part, where the function is defined as a specification of the behaviour between outputs and inputs.

### 4.2.1. Data Collection

* The quantity and quality of your data determines how accurate our model is
* The outcome of this step is usually a representation of the data (Guo simplifies defining a table) that we will use for training
* The use of pre-collected data, like datasets from Kaggle, UCI, etc., also fits in this step

### 4.2.2. Data Preparation

* Wrangle data and schedule, it for training
* Clean up what may be needed (remove duplicates, correct errors, deal with missing values, normalization, data form conversions, etc.)
* Randomize data that erases the effects of a particular order in which we have collected and/or otherwise prepared our data
* Visualize data to help identify important relationships between variables or data.
* There are various algorithms for different tasks; choose the correct one

### 4.2.3. Train the Model

* The training aims to answer a question or make a prediction correctly as often as possible
* An example of linear regression: an algorithm would have to learn values for m (or W) and b(x is input, y is output)
* Each iteration of the cycle is a training step

### 4.2.4. Evaluate the model

* Uses some metric or combination of metrics to assess the objective efficiency of the model• Check the model against previously unseen data
* This unseen data is intended to be somewhat reflective of model performance in the real world, but still helps to refine the model (as compared to measuring data which does not)
* Successful train/eval split? 80/20, 70/30, or equivalent, depending on the area, quality of data, dataset size, etc.

### 4.2.5. Parameter Tuning

* This step refers to hyperparameter tuning, which is an "art form" as opposed to science • Tuning model parameters for improved performance
* Simple model hyperparameters may include multiple training steps, learning rate, initialization and distribution values, etc.

### 4.2.6. Make Predictions

* The use of additional (check set) data that have been omitted from the model (and for which class marks are known) is used to check the model; a better approximation of how the model will work in the real world.

## 4.3. Non – Functional Requirements

Non-functional requirements can be defined as the user’s need for software quality. Addressing a user concern will necessitate the formulation of several functional requirements, but the user concerns will also act to constrain other requirements that are characteristic of nonfunctional requirements.

### 4.3.1. Operation

* **Security:** Cloud provider (Google) level security best practices are implemented for network, routes and authentication
* **Accessibility:** The application has aa intuitive UI and makes it accessible to all kinds of users
* **Availability:** Cloud provider (Google) availability best practices for the application and its services
* **Confidentiality:** User information collected, stored and re-used abide by the terms and conditions set by the company
* **Efficiency:** Conversations are instantaneous and maintain a constant flow.
* **Integrity:** The application data movement is encrypted and any confidential information such as personal details are not exposed to third parties
* **Reliability:** Cloud provider (Google) availability best practices for the application and its services
* **Survivability:** Script deployment and rollout procedures are followed to minimize downtime, frequent disaster recovery and security measures are in place
* **Usability:** The application is helpful to users who have basic knowledge of the English language and its communication essentials

### 4.3.2. Revision

* **Flexibility:** The application is platform-independent as it is a process that runs in the background. Normally the application has the ability for the solution to adapt to possible or future changes in its requirements. Our design built to try to cater to these changes which inevitably arrive in the future. The flexibility is catered for with the design of the system as a whole and the smaller aspects of the system.
* **Maintainability:** The developer operations methods used in the application allows us to easily maintain, fix issues and provide new updates
* **Scalability:** The application can handle any number of requests (practically unlimited) as auto-scaling is enabled for the compute service opted
* **Verifiability:** Continuous integration and testing procedures are incorporated to prevent the application from breaking before being deployed

### 4.3.3. Transition

* **Install-ability:** Since it is a web-based application, it can be used by anyone with access to the internet and a browser
* **Inter-operability:** All the services opted in production are provided by Google Cloud Platform and its services. All the components of the said provider as tightly coupled with each other
* **Portability:** The application is platform-independent and can run on any web browser on any operating system
* **Re-usability:** The application is built as a product to service companies. Hence, multiple people can use the system without affecting other users of the system

# 5. System Design

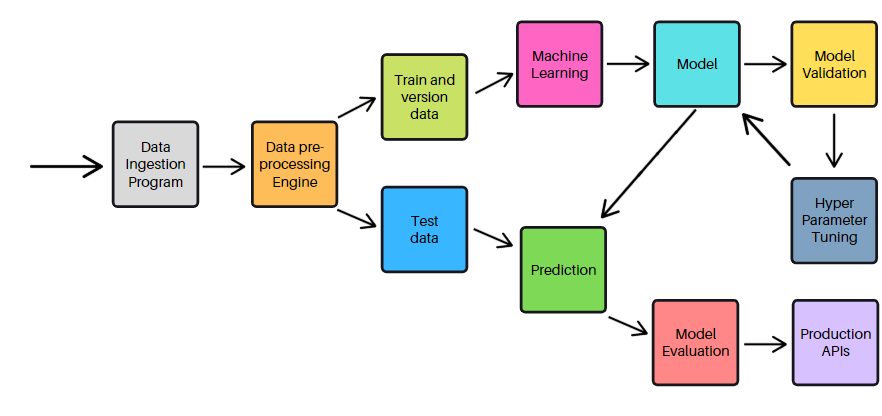
System design is the process of designing the elements of the system, these as the architecture, modules and components, the different interfaces of the devices, and the data that goes through the system. System Analysis is a method that breaks down the system into its parts to decide how well these components operate to satisfy the requirements set. The goal of the system design process is to provide sufficient detailed data and information on the system and its system elements to allow the implementation to be consistent with the architectural entities as defined in the system architecture models and views. System design is a very broad subject. Even a software developer with several years of experience working in a leading IT company may not be an expert in system design. If you want to become an expert, you need to read a lot of books, papers, and solve the real problems of large-scale system design. There is quite a range of use instances where an organization may want an interpretable model. Another use case is when we want to model the task. In such situations, knowing the answer from each source of advertising becomes important. One aspect to emphasize is continuous integration to build accuracy. If the new model is doing better than the old model, why not implement it in production instead of chasing for gradual gains? You can set up an A / B test to test the validity of your argument that your model is better than the current model. Some users (Test Group) will see the one while some users (Control) will see the predictions from the previous model. The aim should just be to will the time for your model's first online experiment. Not only has this generated interest, but it also lets you recognize the shortcomings of your model with real-time input that you can then improve on. Or some other metric, we're always going to want to go to black-box models like NeuralNets or XGBoost. System design is the process of designing the components of the system, such as the software, modules and equipment, the various configurations of the systems, and the data that goes through the system. System Analysis is a process that breaks down the system into its components to determine how well these components work to meet the requirements set. The system design process aims to provide sufficient detailed data and information on the system and its device elements to enable the implementation to be compatible with the architectural entities as specified in the system architecture models and views. The design of the system is a very broad subject. Even a software engineer with many years of experience working in a leading IT company may not be a system design expert.

If you want to be an expert, you need to read a lot of books, journals, and solve the real problems of large-scale system design. There are quite a several circumstances of use where a company might want an interpretable model. One of these uses is when we want to model the task. In such cases, it becomes necessary to know the response from each stream of advertising. One element that needs to be emphasized is continuous integration to improve accuracy. If your current model is performing better than the existing model, why not deploy it in production instead of finding incremental gains? You can set up an A / B test to test the validity of your claim that your model is better than the current one. Some users (Test Group) will see the model while some users (Control) will see the predictions of the previous model. Your goal should always be to reduce the time needed for your model's first online experiment. Not only does this attract interest, but it also lets you understand the shortcomings of your model with real-time feedback that you can then expand on. Or any other metrics, we're either going to want to go to black-box models like NeuralNets or XGBoost.

## 5.1. Architectural Diagram

Traditionally, pipelines include overnight batch processing, i.e. collecting data, sending it through the Business Message Bus and processing to provide pre-calculated results and guidance for next day operations. While this works in some industries, it is insufficient in others, especially when it comes to ML applications.

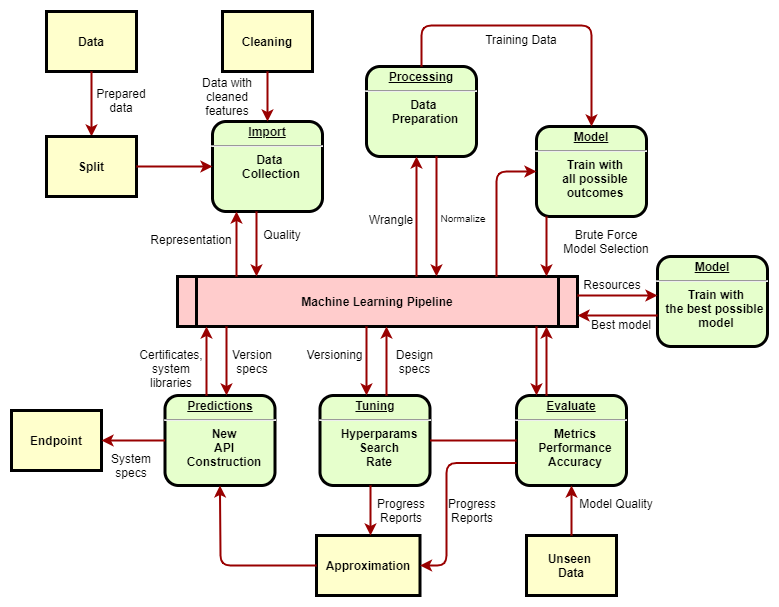
The following diagram (Figure 5.1) illustrates the ML pipeline applied to a real-time business issue where features and forecasts are time-sensitive (e.g. Netflix recommendation engines, Uber arrival time estimation, LinkedIn communication recommendations, Airbnb search engines, etc.).



### Figure 5.1: Architecture of an ML platform

## 5.2. Data Flow Diagram

Dataflow (Figure 5.2) programming languages share some functional features and have typically been developed to add those functional principles to a language that is more suitable for numerical processing. In functional programming, programs are viewed as a series of stateless task evaluations. It is, however, also at the same time a symbolic math library and is also used for machine learning applications such as neural networks. It's an interface for application programming. The Application Program Interface (API) is a series of routines, protocols and resources used to construct software applications.



### Figure 5.2: Data flow of the application

## 5.3. Methodologies

### 5.3.1. Automatic Data Cleaning

Data cleaning is an important aspect of the architecture of the ML pipeline. Data cleaning aims to improve the quality of the data set by removing data errors. Popular error classes include missing values in input data, redundant entries, invalid values or broken relations between multiple data sets entries. In general, data cleaning is divided into two tasks: error detection and error repair. For more than a decade, semi-automatic, interactive systems exist to assist data scientists in the cleaning of data. However, most existing methods still seek to assist a human data scientist rather than fully automated data-cleaning based on a human-defined data quality component, and data-cleaning is handled in a similar way to pipeline structure searching. Basic data-cleaning operators are iteratively combined using greedy searches to create sophisticated data-cleaning. Most of the current AutoML systems recognize the importance of data cleaning and include various phases of data cleaning in the built ML pipeline. However, these data-cleaning steps are typically hardcoded and not created based on some metric during the optimization process. These defined data-cleaning measures typically include imputation of missing values, elimination of samples with incorrect values, such as infinity or outliers, and normalized variety scaling attributes. In general, current AutoML frameworks do not find state-of-the-art data-cleaning science. High criteria for specific data quality are sometimes added at later stages in the ML pipeline, e.g. SVMs need a numerical encoding of categorical features, while random forests can manage them natively. These additional requirements can be defined by evaluating the candidate pipeline and matching the requirements of each stage with the meta-features of each function in the data set. The incorporation of domain knowledge during data cleaning greatly increases the quality of the data. Using various representations of expert knowledge, such as reputation constraints or first-order logic, low-quality data can be identified and corrected automatically. However, these potentials are not used by current AutoML frameworks as they are meant to be fully data-agnostic for a wide range of data sets. As a consequence, the user is given advanced and domain-specific data cleaning.

### 5.3.2. Automatic Feature Engineering

Feature Development is the process of creating and selecting features from a data set for the next modelling stage. This step is crucial for the complete ML pipeline, as the overall model output is highly dependent on the features available. The efficiency of the ML pipeline can be improved several times over the construction of good features (Pyle, 1999). Feature development can be divided into three sub-tasks: extraction of features, design of features and selection of features. Feature engineering— particularly feature creation — is highly domain-specific and very difficult to generalize. Even for one data, iterative feature generation is difficult for a scientist to assess the impact of a feature, as domain expertise is required. As a consequence, feature engineering is mainly a manual and time-consuming job powered by trial and error. Construction is usually aggregated as a feature generation in the context of AutoML feature extraction and feature generation.

### 5.3.3. Feature Selection

Feature Collection selects a subset of the original feature set to speed up the subsequent training of the ML model and improve its performance by removing redundant or misleading apps. Also, the interpretability of the eligible model is increased. An easy domain-agnostic filtering method for the collection of features is based about information theory and statistics. Algorithms such as univariate search, variance threshold, function value, correlation matrices are already incorporated into current AutoML frameworks and selected using conventional CASH methods. More advanced software selection approaches are typically implemented in special feature development frameworks. In general, the feature set — and therefore also its power set — is finite. Software selection by wrapper functions searches for the best subset of features by testing its performance on a particular ML algorithm. Easy approaches use random search or check the power set thoroughly. Heuristic approaches adopt the iterative method by introducing a combination of forwarding and back selection to pick a feature-subset while proposing to model the selection of subsets as a reinforcement problem. Genetic engineering in combination with a low-cost predictive algorithm to achieve a well-performing subset of features.

Finally, there are special feature selection methods that are useful in combination with feature extraction and feature creation. Genetic programming to create new features. Besides, the knowledge on how often each feature was used during the design of the feature is re-used to obtain the importance of the feature. Calculate the meta-features for each new feature, e.g. the diversity of views or the exchange of information with the other features. Using a pre-trained classifier, the impact of a single feature can be expected to pick only promising features.

### 5.3.4. Feature Extraction

Function extraction is a dimensional-reduction method performed by some mapping functions. It extracts insightful and non-redundant functionality based on certain metrics. Unlike the set of features, the extraction function changes the original features. The extraction feature kernel is a mapping function that can be implemented in many ways. Key component analysis (PCA), independent component analysis, iso map, nonlinear dimensional reduction, and linear discriminant analysis (LDA) are the most influential methods. Recently, the feed-forward approach to neural networks has become popular; it uses the hidden units of a pre-trained model as an extracted feature.

### 5.3.5. Model Generation

We need to create a model and set its hyperparameters after generating the features. As model creation consists of two steps: model selection and hyperparameter optimisation. There are two types of model selection approaches traditional model selection and neural architecture search (NAS). The former requires choosing the best-performing model from conventional machine-learning algorithms, such as support vector machine (SVM), k-nearest neighbours (KNN), decision tree, and k-means. This paper focuses more on NAS, which aims to develop a novel neural architecture without human assistance and is currently a very hot topic. To give readers a clear understanding of the NAS methodology, two key aspects of the NAS are introduced: the model structures and the algorithms used to optimize the parameters of the created model (i.e. the hyperparameter optimization algorithms).

The model is created by selecting and combining a set of primitive operations that are predefined in the search space. Operations can be generally divided into categories such as convolution, pooling, concatenation, elementary addition, and skip link. Empirically, these 5 well-designed operations can help improve the performance of the generated model; thus, many human-designed modules are used as operations, such as deep-wise separable convolution, dilated convolution. The parameters of these operations are also, as a general rule, empirically predefined. For example, the kernel size of the convolution is usually set to 3×3 and 5×5. Model structures are summarized as follows: the first and most straightforward way to create a model structure is to generate an whole chain-structured neural network, providing two simplistic examples of the entire chain-structured models generated, which are constructed by stacking a predefined number of nodes, where each node represents a single layer and has a specified operation. The simplest structure is the left model, while the right model is more complicated, as it allows arbitrary skip connections to exist between the ordered nodes, as these connections have proven to be effective in practice. Although the entire structure is easy to implement, it has a number of disadvantages. For example, it is widely accepted that the deeper the model, the better the generalization potential, but the more costly the search for such a deep network is. In addition, the generated architecture lacks transferability: i.e. a model generated on a small dataset may not match a larger dataset. It involves the generation of a new model for a larger data set.

### 5.3.6. Algorithm Selection and Hyperparameter Optimization

Let the shape g help G, the loss function L and the training set D be given. The algorithm must be selected and configured through hyperparameters for each node in g. This section discusses various methods for the collection and configuration of algorithms. The notion first proposed and since then adopted by many others is the problem of combination algorithm selection and hyperparameter optimization (CASH). Instead of first choosing an algorithm and then optimizing its hyperparameters, both steps are executed simultaneously. This problem is conceived as a black-box optimization problem that leads to a minimization problem that is quite similar to the pipeline formation problem. 5-007r represents the root node with the child node for each algorithm. According to the mandatory hyperparameters as child nodes, all conditional hyperparameters are children with one mandatory hyperparameter.

This tree structure can be used to reduce search space significantly. The rest of this section introduces many optimization techniques to solve the problem.

#### 5.3.6.1. Grid Search

The first method suggested to systematically explore the configuration field was the quest for a grid. As the name implies, the grid search produces a list of configurations and checks them all. While grid search is easily implemented and parallelized, it has two major drawbacks: 1) it does not scale well for wide configuration spaces, as the number of function evaluations increases exponentially with the number of hyperparameters, and 2) the hierarchical hyperparameter structure is not considered, leading to redundant configurations. In the classic version, grid quest does not take advantage of the knowledge of well-performing regions. This drawback is partially eliminated by counteracting the grid search. At first, a large grid is mounted, then a finer grid is built centred around the most effective configuration. This iterative procedure is repeated k times at a local minimum.

#### 5.3.6.2. Random Search

Another well-known solution is random search. Candidate configuration is created by randomly selecting a value for each hyperparameter independently of all others. Conditional hyperparameters can be implicitly managed through the hierarchical dependency network. Random search is easy to implement and parallelize and is well suited for gradient-free functions with many local minima. Although the convergence speed is faster than the grid search, many functional evaluations are still required as no awareness of well-performing regions is exploited. Since the estimation of functions is very costly, a random search requires a long period of optimization.

#### 5.3.6.3. Sequential Model-Based Optimization

The CASH problem can be viewed since a regression problem: the loss method can be approximated using standard regression methods based on the hyperparameter configurations that have been tested so far. This principle is embodied by sequential model-based optimisation. Such samples are used to construct a regression model for the objective function. Next, a new configuration is selected and tested using the objective function. Finally, a new tuple is applied to the sample collection to sequentially create new configurations. These new configurations are obtained using a low-cost discovery function. -the proposed configuration is evaluated on the objective function f and the result is added. These steps are repeated until they have been fixed. Either a set number of iterations or a time limit—is typically exhausted. The initialisation is mostly achieved by choosing a small number of random configurations. While fitting a one and selecting a configuration imposes a computational overhead, the likelihood of testing a malfunctioning configuration can be greatly reduced. Since the actual test feature is usually much more costly than such intermediate steps, better configurations can be identified in a shorter period compared to random or grid searches. Bayesian optimization is used to implement the replacement model fitting and configuration selection. Bayesian optimization is an iterative optimization technique that is well adapted to expensive objective functions. Bayesian optimization is very effective concerning the number of objective function evaluations, as the acquisition function is used to assess the next configuration ~5-007n+1 to be evaluated. The acquisition function automatically handles the trade-off between exploration and exploitation: new regions with a large level of uncertainty are explored, preventing optimization from sticking to a local minimum; well-performing regions with a low level of uncertainty are exploited, converging to a local minimum. The surrogate model M is the same as the posterior model. Since mentioned above, the characteristics and form of the loss function are generally unknown. The posterior must, therefore, be a non-parametric model. Gaussian processes are the standard surrogate model for Bayesian optimization. The main concept is that every objective function f can be modelled using an infinite-dimensional Gaussian distribution. The main downside of Gaussian processes is the complexity of runtime, as testing a high number of configurations is prohibitively expensive. The insufficient native support for categorical input1 and the use of the search space structure is a more important downside for CASH. Random forest regression is a mixture of multiple regression trees.

Regression trees use recursive splitting of training data to produce groups of related observations. Apart from being able to manage categorical variables natively, random forests are easy to train and even faster to test new data when obtaining strong predictive power. As far as a tree structure is concerned, TPE natively handles hierarchical search spaces by modelling -hyperparameter individually. These distributions are hierarchically linked, reflecting the dependences between the hyperparameters resulting in a pseudo multidimensional distribution.

#### 5.3.6.4. Evolutionary Algorithms

The evolutionary algorithms are an alternative to SMBO. Evolutionary algorithms are a series of different population-based optimization algorithms based on biological evolution. In general, evolutionary algorithms are applied to a wide range of optimization problems, as no assumptions about the objective function are needed.

#### 5.3.6.5. Multi-Armed Bandit Learning

Many SMBO methods have mixed and conditional search space. By conducting a grid search with only categorical hyperparameters, the configuration space can be divided into a finite set of smaller configuration spaces — called a hyper partition — with only continuous hyperparameters. Each hyper partition can be optimized using Bayesian standard optimization methods. The selection of a hyper partition can be modelled as a multi-armed bandit problem. While multi-armed bandit learning can also be extended to continuous optimization, it is only used in a finite setting in conjunction with other optimization techniques in the sense of AutoML.

#### 5.3.6.6. Gradient Descent

The gradient descent, an iterative minimization algorithm, is a very strong optimization tool. If f is distinguished and its closed-form representation is known, the gradient is computable.

However, the closed-form representation of f for CASH is not known and therefore gradient descent is usually not applicable. By assuming certain properties of f — and thus restricting the applicability of this approach to a specific problem instance— gradient descent can still be used. Owing to strict constraints, gradient descent is not further studied.

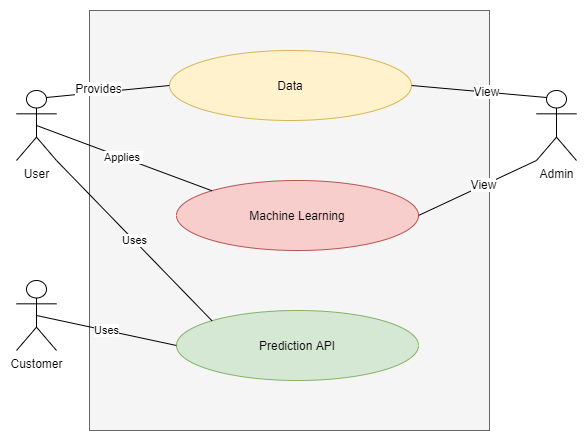
### 5.3.7 Post-Modelling

Provides complete ecosystem data collection, processing, repair, integration, discovery, query, visualization, and analytics with the Restful API. Enables users to share data, interact with others, and perform a wide range of operations from the available suite of data processing methods. However, there are many attempts to synthesize automated applications from the available data features. Automated feature engineering that follows a deep feature synthesis algorithm that can work with relational databases making use of an object, forward, and the backward relationship of tables to generate new higher-level features iteratively that can enhance modelling performance. In reality, there is an immediate need to try to incorporate the best algorithms and tools in different phases of the process into a single workflow. This move will be the corner-stone of the replacement of the data scientist. It was launched as an open-source framework to handle the machine learning pipeline from the end to the end. It is a language-agnostic framework that has a REST API and command-line interface in addition to the APIs for the most common programming languages such as Python, R, and Java. The project executes three different operations: recording results from experiments and workflows from various applications and algorithms. Besides, code versions with the metrics used, parameter settings, and visualizations made can all be tracked and stored. Bundle of the code used in the reusable and reproducible format chain to be shared with the entire community or transferred directly to the output. Over and above, this handles all the appropriate dependencies and entry points. Handle and deploy models built from various workflows across a wide range of platforms. It allows a variety of activities for the data scientist. However, there is still a lack of smartness in determining the best workflows that are suitable for each role and require human intervention in taking multiple actions and resolving conflicts that occur through the transition of models between different platforms. It also supports real-time visualization for monitoring the training process with different metrics and parameters.

Python, a software model management framework that lets data scientists store logs and outcomes experiments with easy reproducibility and project versioning. It also enables synchronization between these stored logs with user private cloud storage files.

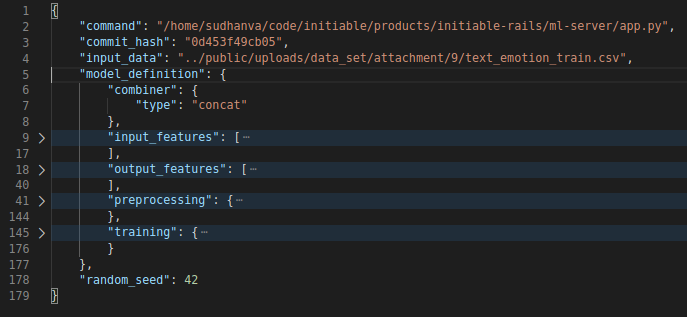
## 5.4. Use Case Diagram

The most basic use case diagram is a description of the interaction of the user with this project, which illustrates the relationship between the user and the various use cases in which the user is involved. The use case diagram will describe the different types of device users and the various use cases and will often be followed by other types of diagrams. Use cases are represented either by circles or ellipses.

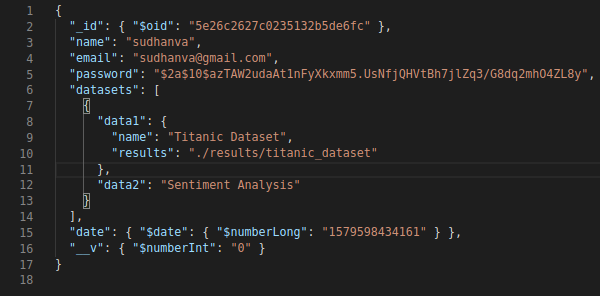


### Figure 5.3: Use Case Diagram

## 5.5. Document Structure



### Figure 5.4: Document Structure for the database during machine learning



### Figure 5.5: Document Structure for the database during data ingestion

# 6. Implementation

## 6.1. Sample Source Code

### 6.1.1 Data set controller – Rails

|  |
| --- |
| class DataSetsController < ApplicationController |
| before\_action :set\_data\_set, only: [:show, :edit, :update, :destroy] |
| before\_action :authenticate\_user! |
|  |
| # GET /data\_sets |
| # GET /data\_sets.json |
| def index |
| @data\_sets = DataSet.all |
| end |
|  |
| # GET /data\_sets/1 |
| # GET /data\_sets/1.json |
| def show |
| @data\_set = DataSet.find(params[:id]) |
| require 'csv' |
|  |
| csv\_text = File.read(Rails.public\_path.to\_s + "" + @data\_set.attachment\_url) |
| csv = CSV.parse(csv\_text, :headers => true) |
| @csv = csv |
| end |
|  |
| def results |
| @data\_set = params |
| end |
|  |
| # GET /data\_sets/new |
| def new |
| @data\_set = DataSet.new |
| end |
|  |
| # GET /data\_sets/1/edit |
| def edit |
| end |
|  |
| # POST /data\_sets |
| # POST /data\_sets.json |
| def create |
| @data\_set = DataSet.new(data\_set\_params) |
|  |
| respond\_to do |format| |
| if @data\_set.save |
| format.html { redirect\_to @data\_set, notice: 'Data set was successfully created.' } |
| format.json { render :show, status: :created, location: @data\_set } |
| else |
| format.html { render :new } |
| format.json { render json: @data\_set.errors, status: :unprocessable\_entity } |
| end |
| end |
| end |
|  |
| # PATCH/PUT /data\_sets/1 |
| # PATCH/PUT /data\_sets/1.json |
| def update |
| respond\_to do |format| |
| if @data\_set.update(data\_set\_params) |
| format.html { redirect\_to @data\_set, notice: 'Data set was successfully updated.' } |
| format.json { render :show, status: :ok, location: @data\_set } |
| else |
| format.html { render :edit } |
| format.json { render json: @data\_set.errors, status: :unprocessable\_entity } |
| end |
| end |
| end |
|  |
| # DELETE /data\_sets/1 |
| # DELETE /data\_sets/1.json |
| def destroy |
| @data\_set.destroy |
| respond\_to do |format| |
| format.html { redirect\_to data\_sets\_url, notice: 'Data set was successfully destroyed.' } |
| format.json { head :no\_content } |
| end |
| end |
|  |
| private |
| # Use callbacks to share common setup or constraints between actions. |
| def set\_data\_set |
| @data\_set = DataSet.find(params[:id]) |
| end |
|  |
| # Only allow a list of trusted parameters through. |
| def data\_set\_params |
| params.require(:data\_set).permit(:name, :attachment, :attachment\_test) |
| end |
| end |

### 6.1.2 Data set model – Rails

Top of Form

Bottom of Form

Top of Form

Bottom of Form

|  |
| --- |
| class DataSet < ApplicationRecord |
| mount\_uploader :attachment, AttachmentUploader # Tells rails to use this uploader for this model. |
| mount\_uploader :attachment\_test, AttachmentUploaderTest # Tells rails to use this uploader for this model. |
| validates :name, presence: true # Make sure the owner's name is present. |
| end |

### 6.1.3. ML Server – Python/Flask – Model Definition

|  |
| --- |
| class InitiableModel: |
|  |
| def \_\_init\_\_( |
| self, |
| model\_definition=None, |
| model\_definition\_file=None, |
| logging\_level=logging.ERROR |
| ): |
|  |
| if model\_definition is None and model\_definition\_file is None: |
| raise ValueError( |
| 'Either model\_definition of model\_definition\_file have to be' |
| 'not None to initialize a InitiableModel' |
| ) |
|  |
| if model\_definition is not None and model\_definition\_file is not None: |
| raise ValueError( |
| 'Only one between model\_definition and ' |
| 'model\_definition\_file can be provided' |
| ) |
|  |
| self.set\_logging\_level(logging\_level) |
|  |
| if model\_definition\_file is not None: |
| with open(model\_definition\_file, 'r') as def\_file: |
| self.model\_definition = merge\_with\_defaults( |
| yaml.safe\_load(def\_file) |
| ) |
| else: |
| model\_definition\_copy = copy.deepcopy(model\_definition) |
| self.model\_definition = merge\_with\_defaults(model\_definition\_copy) |
|  |
| self.train\_set\_metadata = None |
| self.model = None |
| self.exp\_dir\_name = '' |
|  |
| @staticmethod |
| def set\_logging\_level(logging\_level): |
| logging.getLogger('initiable').setLevel(logging\_level) |
| if logging\_level in {logging.WARNING, logging.ERROR, logging.CRITICAL}: |
| set\_disable\_progressbar(True) |
|  |
| @staticmethod |
| def \_read\_data(data\_csv, data\_dict): |
| if data\_csv is not None: |
| data\_df = read\_csv(data\_csv) |
| elif data\_dict is not None: |
| data\_df = pd.DataFrame(data\_dict) |
| else: |
| raise ValueError( |
| 'No input data specified. ' |
| 'One of data\_df, data\_csv or data\_dict must be provided' |
| ) |
|  |
| return data\_df |
|  |
| @staticmethod |
| def load(model\_dir): |
| #-- |
|  |
| model, model\_definition = load\_model\_and\_definition(model\_dir) |
| initiable\_model = InitiableModel(model\_definition) |
| initiable\_model.model = model |
| initiable\_model.train\_set\_metadata = load\_metadata( |
| os.path.join( |
| model\_dir, |
| TRAIN\_SET\_METADATA\_FILE\_NAME |
| ) |
| ) |
| return initiable\_model |
|  |
| def save(self, save\_path): |
| #-- |
| if (self.model is None or self.model.session is None or |
| self.model\_definition is None or self.train\_set\_metadata is None): |
| raise ValueError('Model has not been initialized or loaded') |
|  |
| model\_weights\_path = os.path.join(save\_path, MODEL\_WEIGHTS\_FILE\_NAME) |
|  |
| model\_hyperparameters\_path = os.path.join( |
| save\_path, |
| MODEL\_HYPERPARAMETERS\_FILE\_NAME |
| ) |
|  |
| self.model.save\_weights(self.model.session, model\_weights\_path) |
|  |
| train\_set\_metadata\_path = os.path.join( |
| save\_path, |
| TRAIN\_SET\_METADATA\_FILE\_NAME |
| ) |
| save\_json(train\_set\_metadata\_path, self.train\_set\_metadata) |
|  |
| self.model.save\_hyperparameters( |
| self.model.hyperparameters, |
| model\_hyperparameters\_path |
| ) |
|  |
| def save\_for\_serving(self, save\_path): |
| if (self.model is None or self.model.session is None or |
| self.model\_definition is None or self.train\_set\_metadata is None): |
| raise ValueError('Model has not been initialized or loaded') |
|  |
| self.model.save\_savedmodel(save\_path) |
|  |
| def close(self): |
| #-- |
| if self.model is not None: |
| self.model.close\_session() |
|  |

### 6.1.4. ML Server – Python/Flask – Train Method

|  |
| --- |
| def train( |
| self, |
| data\_df=None, |
| data\_train\_df=None, |
| data\_validation\_df=None, |
| data\_test\_df=None, |
| data\_csv=None, |
| data\_train\_csv=None, |
| data\_validation\_csv=None, |
| data\_test\_csv=None, |
| data\_hdf5=None, |
| data\_train\_hdf5=None, |
| data\_validation\_hdf5=None, |
| data\_test\_hdf5=None, |
| data\_dict=None, |
| data\_train\_dict=None, |
| data\_validation\_dict=None, |
| data\_test\_dict=None, |
| train\_set\_metadata\_json=None, |
| experiment\_name='api\_experiment', |
| model\_name='run', |
| model\_load\_path=None, |
| model\_resume\_path=None, |
| skip\_save\_training\_description=False, |
| skip\_save\_training\_statistics=False, |
| skip\_save\_model=False, |
| skip\_save\_progress=False, |
| skip\_save\_log=False, |
| skip\_save\_processed\_input=False, |
| output\_directory='results', |
| gpus=None, |
| gpu\_fraction=1.0, |
| use\_horovod=False, |
| random\_seed=42, |
| debug=False, |
| \*\*kwargs |
| ): |
|  |
| if data\_df is None and data\_dict is not None: |
| data\_df = pd.DataFrame(data\_dict) |
|  |
| if data\_train\_df is None and data\_train\_dict is not None: |
| data\_train\_df = pd.DataFrame(data\_train\_dict) |
|  |
| if data\_validation\_df is None and data\_validation\_dict is not None: |
| data\_validation\_df = pd.DataFrame(data\_validation\_dict) |
|  |
| if data\_test\_df is None and data\_test\_dict is not None: |
| data\_test\_df = pd.DataFrame(data\_test\_dict) |
|  |
| ( |
| self.model, |
| preprocessed\_data, |
| self.exp\_dir\_name, |
| train\_stats, |
| self.model\_definition |
| ) = full\_train( |
| self.model\_definition, |
| data\_df=data\_df, |
| data\_train\_df=data\_train\_df, |
| data\_validation\_df=data\_validation\_df, |
| data\_test\_df=data\_test\_df, |
| data\_csv=data\_csv, |
| data\_train\_csv=data\_train\_csv, |
| data\_validation\_csv=data\_validation\_csv, |
| data\_test\_csv=data\_test\_csv, |
| data\_hdf5=data\_hdf5, |
| data\_train\_hdf5=data\_train\_hdf5, |
| data\_validation\_hdf5=data\_validation\_hdf5, |
| data\_test\_hdf5=data\_test\_hdf5, |
| train\_set\_metadata\_json=train\_set\_metadata\_json, |
| experiment\_name=experiment\_name, |
| model\_name=model\_name, |
| model\_load\_path=model\_load\_path, |
| model\_resume\_path=model\_resume\_path, |
| skip\_save\_training\_description=skip\_save\_training\_description, |
| skip\_save\_training\_statistics=skip\_save\_training\_statistics, |
| skip\_save\_model=skip\_save\_model, |
| skip\_save\_progress=skip\_save\_progress, |
| skip\_save\_log=skip\_save\_log, |
| skip\_save\_processed\_input=skip\_save\_processed\_input, |
| output\_directory=output\_directory, |
| should\_close\_session=False, |
| gpus=gpus, |
| gpu\_fraction=gpu\_fraction, |
| use\_horovod=use\_horovod, |
| random\_seed=random\_seed, |
| debug=debug, |
| ) |
|  |
| self.train\_set\_metadata = preprocessed\_data[-1] |
|  |
| return train\_stats |

### 6.1.5. Authentication – Node.js

|  |
| --- |
| const express = require('express'); |
| const router = express.Router(); |
| const bcrypt = require('bcryptjs'); |
| const jwt = require('jsonwebtoken'); |
| const config = require('config'); |
| const auth = require('../middleware/auth'); |
| const { check, validationResult } = require('express-validator'); |
|  |
| const User = require('../models/User'); |
|  |
| // @route GET api/auth |
| // @desc Get logged in user |
| // @access Private |
| router.get('/', auth, async (req, res) => { |
| console.log(req.user) |
| try { |
| const user = await User.findById(req.user.id).select('-password'); |
| res.json(user); |
| } catch (err) { |
| console.error(err.message); |
| res.status(500).send('Server Error'); |
| } |
| }); |
|  |
| // @route POST api/auth |
| // @desc Auth user & get token |
| // @access Public |
| router.post( |
| '/', |
| [ check('email', 'Please include a valid email').isEmail(), check('password', 'Password is required').exists() ], |
| async (req, res) => { |
| const errors = validationResult(req); |
| if (!errors.isEmpty()) { |
| return res.status(400).json({ errors: errors.array() }); |
| } |
|  |
| const { email, password } = req.body; |
|  |
| try { |
| let user = await User.findOne({ email }); |
|  |
| if (!user) { |
| return res.status(400).json({ msg: 'Invalid Credentials' }); |
| } |
|  |
| const isMatch = await bcrypt.compare(password, user.password); |
|  |
| if (!isMatch) { |
| return res.status(400).json({ msg: 'Invalid Credentials' }); |
| } |
|  |
| const payload = { |
| user: { |
| id: user.id |
| } |
| }; |
|  |
| jwt.sign( |
| payload, |
| config.get('jwtSecret'), |
| (err, token) => { |
| if (err) throw err; |
| res.json({ token }); |
| } |
| ); |
| } catch (err) { |
| console.error(err.message); |
| res.status(500).send('Server Error'); |
| } |
| } |
| ); |
|  |
| module.exports = { |
| authController: router |
| }; |

## 6.2. Screenshots

### Screenshot 6.1: Listing all datasets for a particular user

### Screenshot 6.2: Listing all features (feature engineering) of a dataset of a particular user

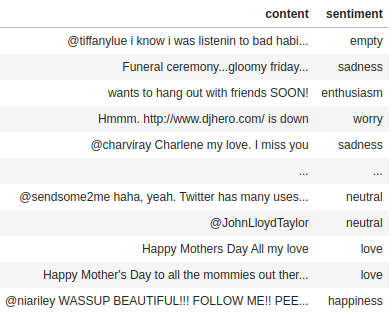
### Screenshot 6.3: Prediction APIs that sends back a response based on a particular request

### Screenshot 6.4: Docker Microservices running

### Screenshot 6.5: Authentication Token Verification

# 7. MODEL EVALUATION AND PERFORMANCE

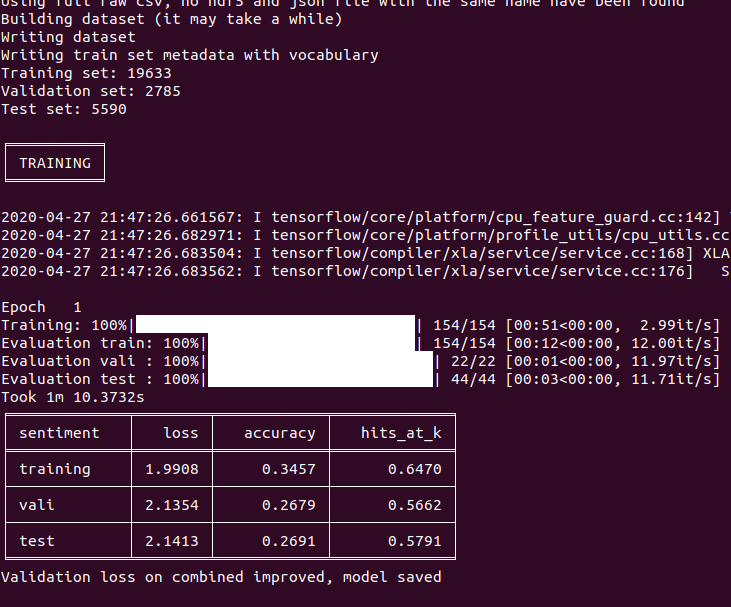
As an example, a dataset of sentiments is used. This data set consists of a column of *content* and a column of *sentiment*. *Content* represents the text and the *sentiment* is a categorical variable representing sentiments. Below (Figure 7.1) is a gist at the real dataset being used here for machine learning.



## Figure: 7.1: Sample Dataset used in this example

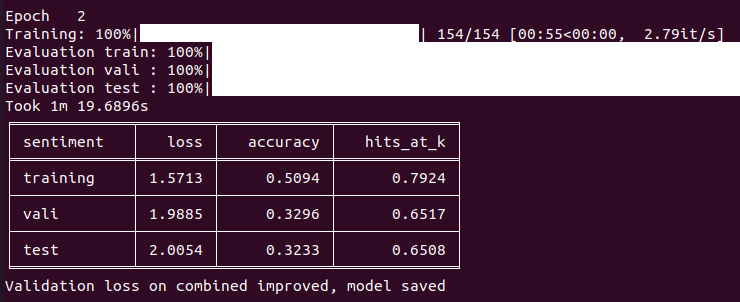
The codebase is structured in a modular, data form / function-centric manner so that adding a new data type feature is quite easy and involves independent code changes. All different logic datatypes are stored in the corresponding function module. Function classes contain raw data preprocessing logic unique to each type of data.

All input (output) features follow the create input (build performance) method that is used to construct encodings (decode outputs). Output features also include data type-specific logic for calculating output measurements such as loss, precision, etc. Encoders and decoders are modularized so that different features can be used. For example, sequence encoders are shared between text, sequence and time-series features. Various Model Architecture Components that can be reused are also divided into dedicated modules, e.g. convolution modules, fully connected modules, etc. The bulk of the training logic resides in model.py, which initializes a TensorFlow session, feeds data and conducts training.



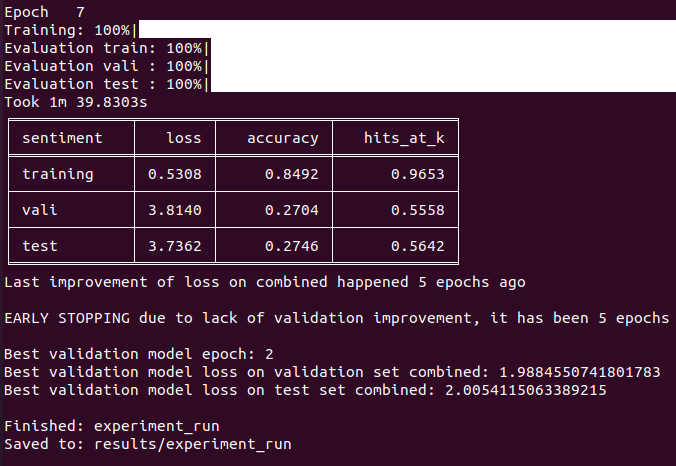
## Figure: 7.2: Model Evaluation - Initial Stages

Selection of models that have been educated in the selection, including 5-fold cross-validated model results (by default). The number of folds used in the model assessment process can be modified using the folds parameter. If the user wants to score models on a particular dataset, they may assign a framing statement, and the model will display scores on that dataset instead. Models are rated by default based on the problem form (second column of the leaderboard). In the case of binary classification problems, that metric is AUC, and in the case of multiclass classification problems (such as this example Figure 7.3), the metric is a mean error per class. In regression problems, the default metric is deviance. For convenience, some additional metrics are also given. It also shows the number of epochs, training losses, validation losses and test losses.



## Figure: 7.3: Model Evaluation – Sample Epoch

After the model reaches its final epoch, due to lack of validation improvement (Figure: 7.4), it concludes the training process and picks the best model out of the trained epochs.



## Figure: 7.3: Model Evaluation – Final Epoch

After training the model, the test dataset is used to evaluate the quality and accuracy of the new model and to provide an aggregate collection of performance metrics to indicate how well the model worked on the test dataset. Using the assessment criteria to assess the quality of your model depends on your business needs and the problem the model is equipped to solve. For example, false positives may cost more than false negatives, or vice versa. If you include a weight column in your training data, the evaluation metrics will not be affected. Weights are only considered during the training phase. Classification models shall provide the following metrics:

## 7.1. Evaluation metrics for classification models

* **AUC PR**: Area under the Precision-Recall (PR) Curve. Its value varies from zero to one, where a higher value implies a higher quality standard.
* **AUC ROC:** Area under the Receiver Operating Characteristics (ROC) curve. This ranges from zero to one, where a higher value indicates a higher quality model.
* **Accuracy:** The fraction of the classification predictions produced by the correct model.
* **Log loss:** A cross-entropy between the model prediction and the target values. This ranges from zero to infinity, where a lower value indicates a higher quality model.
* **F1 score:** The harmonic mean of precision and recall. F1 is a useful metric if you're looking for a balance between precision and recall and an uneven distribution of class.
* **Precision:** The fraction of the classification predictions generated by the correct model.

* **Recall:** The fraction of rows with this mark that the model had correctly predicted. Also referred to as "True positive rate."
* **False-positive rate:** The fraction of rows predicted by the model to be the target label but not (false positive).

These metrics are returned for each separate value of the target column. In the case of multi-class classification models, these metrics are micro-averaged and returned as summary metrics. The metrics for the minority class are used as summary metrics for binary classification models. Micro-averaged metrics are the expected value of each metric in a random sample from your dataset. In addition to the above metrics, there are two other ways to consider your classification model, the uncertainty matrix and the function value graph.

## 7.2. Evaluation metrics for regression models

* **MAE:** The mean absolute error (MAE) is the mean absolute difference between the target values and the expected values. It varies from zero to infinity; a lower value implies a higher quality standard.
* **RMSE:** The root-mean-square error equation is a widely used calculation of the variations between the values predicted by the model or estimator and the values observed. This ranges from zero to infinity; a lower value indicates a higher quality standard.
* **RMSLE:** The root-mean-square logarithmic error metric is similar to the RMSE, except that it uses the normal logarithm of the predicted and actual values plus 1. RMSLE penalizes under-prediction more than over-prediction. It can also be a good metric if you don't want to penalize differences more harshly for large predictive values than for small predictive values. This ranges from zero to infinity; a lower value indicates a higher quality model. The RMSLE evaluation metric is returned only if all the labels and predicted values are non-negative.
* **MAPE:** The mean absolute percentage error (MAPE) is the mean absolute percentage difference between the labels and the predicted values. This metric ranges from zero to infinity; a lower value indicates a higher quality model.

* **Feature Importance:** It informs you what features it considers to be most important for building its model in the Feature Significance Map. Feature importance is determined by calculating the effect that each feature has on the prediction when it is disrupted across a large range of values sampled from the dataset. You must review this information to ensure that all of the most important features make sense of your data.

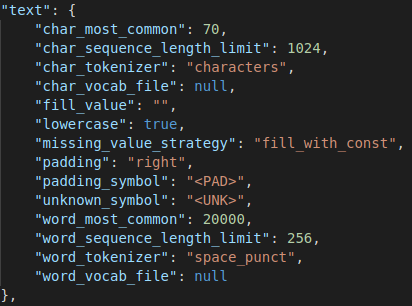
# 8. RESULT AND DISCUSSION

After the model is created, it is deployed as a production API where the APIs can be used for real-time predictions. It accepts one row of data and provides a predicted result based on your model for that data. You can use predictions when you need a prediction as input for your business logic flow.

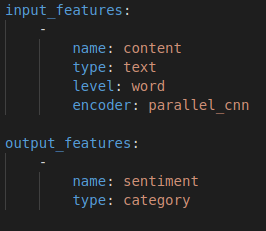
## 8.1 Evaluate JSON and Feature Importance

It tells you what features it considers to be most important for building this model in the Chart of Value. The function value is calculated by measuring the impact that each function has on the prediction when it is disturbed over a wide range of values sampled from the dataset. You must review this information to ensure that all of the most important features make sense of your data. Micro-averaged accuracy is determined by adding together the number of true positive (TP) for each potential value of the target column and dividing it by the number of true positive (TP) and true negatives (TN) for each potential value. A score threshold is a number which ranges from 0 to 1. It provides a way to determine the minimum level of confidence where the value of the prediction should be taken as true.

For example, if you have a class that is unlikely to be the actual value, you would want to lower the threshold for that class; using a threshold of 5 or higher would make that class extremely rarely (or never) expected. The higher threshold reduces false positives, at the cost of more false negatives. The lower threshold reduces false negatives at the cost of more false positives. Put another way, the score threshold affects accuracy and recall. Higher threshold increase accuracy (because the model never predicts unless it is extremely certain) but decreases recall (the percentage of positive examples that the model is correct). Below (Figure 8.1) shows the importance of a function for the sample dataset selected. And Figure 8.2 indicates the features selected by the customer.



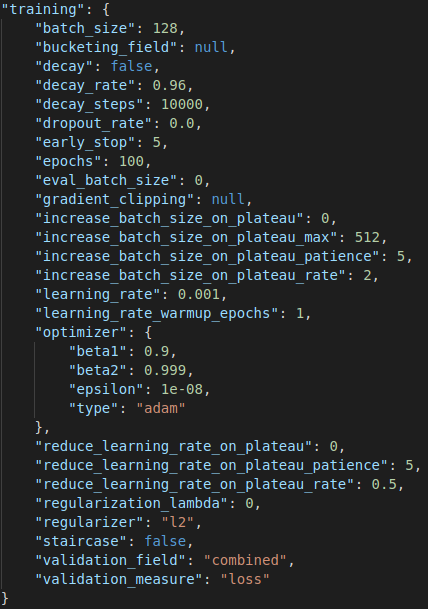
## Figure: 8.1: Text Feature Importance



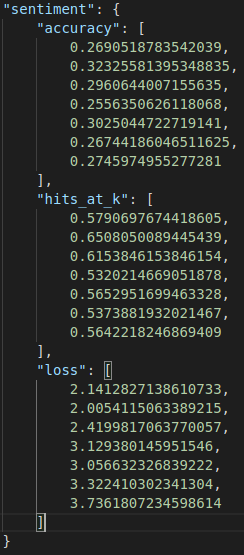
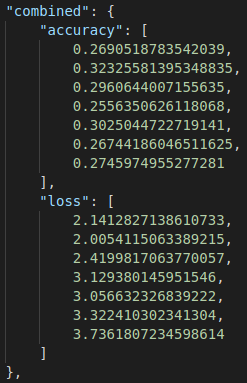
## Figure: 8.2: Text Feature Importance

Creating many more features and engineering by combining them is just the first step in building a good ML model. If we try to generate a set of new features based on just a few technical rules and finitely combine them, it will quickly lead to a combinatorial explosion of possible features to be used. Moreover, most of the new features are not very useful. We need a strategy to eliminate unnecessary features and work with just a subset that can have the most impact on our ML models. Fortunately, many of these strategies can be automated and implemented in a systematic programmatic manner.

Here's an example of a training course:

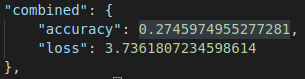


## Figure: 8.3: Training Parameters



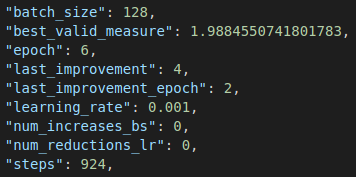
## Figure: 8.4: Accuracy and Loss – Combined and Separated

Selecting features based on the value of some test model. We build a model, not necessarily the best one for a given problem, and then look at the value of that model. For example, we can look at the absolute values of the linear regression coefficients provided by the tree-based model. We may set some threshold and delete all the functions that fall below it. Forward collection and/or recursive deletion of features. In this method, we add one-by-one features to the model and retain only those for which the model improves or remove functions and keep only those for which the model deteriorates. Permutation effect: we change the values of the features, one by one. A function is important if the shuffling (permuting) values increase the predictive error and are unimportant if the shuffling does not have an impact on the prediction error.

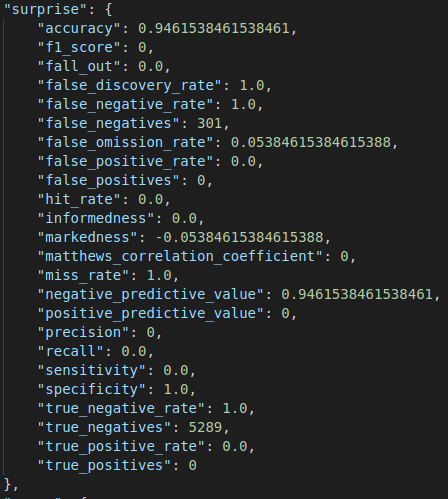


## Figure: 8.5: Test Accuracy and Loss

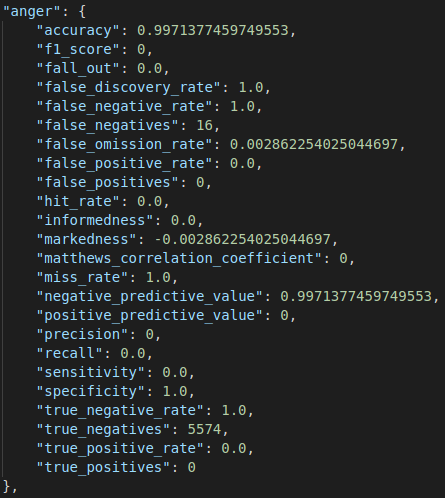
Function collection of genetic algorithms. Genetic algorithms are based on their biological counterparts. This approach creates a lot of randomly shuffled features and tests their "fitness" based on how well they support the ML algorithm. The process is repeated several times, each equivalent to a step in natural evolutionary selection. After many of these measures, only the "fittest" genes (sets of features) can "survive."



## Figure: 8.6: Test parameters and final model parameters



## Figure: 8.7: Sample sentiment parameters - *Surprise*



## Figure: 8.8: Sample sentiment parameters - *Anger*

# 9. Testing

## 9.1. Continuous Integration

Agile and DevOps continue to spread to IT ventures at a rate we've never seen before. As interest in these issues increases, there is also an increase in the set of processes and tools that make it possible to deliver better applications. The "Phase" execution is at the end of the software development lifecycle. And if delivery is not swift, the entire Agile development cycle may be disrupted. Modern software should be designed to be distributed easily, using suitable automation tools. Cloud Build is a fully managed Google Cloud Platform service that lets you build software quickly across all languages, relying on Docker to get the job done. Code testing is a way to check the system to differentiate any blunders, gaps or missing pre-requisites from the real necessity. Programming testing is comprehensively divided into two types-utilitarian testing and non-useful testing. At the point of starting the test exercises: testing should be started as timely as it is possible to reduce the cost and time to rewrite and build code that is bug-free so that it continues to be passed on to the consumer. However, in the Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering stage and continued until the product is out in the preparations. It also relies on the model of advancement that is being used. For example, in the Waterfall model, testing begins at the test stage which is very low in the tree; but in the V-model, testing is carried out parallel to the progress stage.

## 9.2. Unit Testing

Unit Testing is the programming evaluation aspect where the individual units/parts of the software are evaluated. The reason for the existence is to allow the execution of each unit of the product as anticipated. The unit is the smallest testable piece of any object. More often than not, it has one or a few sources of information and, as a consequence, a single yield. Through procedural programming, the unit could be an individual program, function, process, and so on. The smallest unit in the programmed element is a tactic that may have a position with a base/superclass, a special class or a determined / youngster class. (Some view the application module as a product.

This is to be undermined as there will most likely be a large number of individual units inside the module). Unit testing devices, drivers, stubs, and counterfeit/counterfeit products are used for unit testing.

### 9.2.1. Unit Testing Benefits:

Device testing builds confidence in modifying / searching for code. On the one possibility the major device tests will be written and on the off chance that they will be run any time the code is changed, we will most likely automatically get to know any imperfections due to the change. Likewise, because, as of now, codes are less associated to allow unit testing possible, the unintended impact of any code changes is less. Codes are becoming rapidly reusable. Codes should be omitted to make unit testing potential. This means that codes are easier to reuse. Production is quicker than that. On the chance that you don't have a unit testing setup, you're composing your code; (you set up some breakpoints, fire up the GUI, send a few data sources that hopefully hit your code and expect you to be on the right track.) But, if you have a unit testing set up, you're writing the test, composing the code, and running the test. Composing the tests requires a substantial investment, however, the time is remunerated by the less time it takes to run the tests; you do not need to start the GUI and have each of these sources of information. What more unit tests are more reliable than model tests. Advancement is also faster over the long haul. The energy needed to discover and repair the dropouts discovered in the mid-unit test is less in contrast to the effort required to correct the absconds discovered in the device test or recognition test. The cost of correcting an imperfection detected in the unit check is lower relative to the cost of deformities reported in higher amounts. Look at the expense (time, effort, obliteration, embarrassment) of an imperfection that has been identified in the recognition test or when the product is alive.

### 9.2.2. Unit Test Cases

### Table 9.1 shows the list of unit test cases that will be run by continuous integration

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case** | **Scenario** | **Input** | **Expected Result** | **Actual Result** | **Result** |
| **TC01** | **POST** /users | Checking authorization | Email ID | Log In | Logged In | PASS |
| **TC02** | **POST** /users | Checking authorization | Wrong Email ID | Error | Error: Wrong Email | PASS |
| **TC03** | **POST** /users  /sign\_in | Authentication | AdminToken | Unique ID | Unique IDgenerated successfuly | PASS |
| **TC04** | **POST** /users  /sign\_in | Authentication | Different AdminToken | Duplicate  ID | Error: Wrong auth token | PASS |
| **TC05** | **GET** /datasets | Retrieve stored user data | GET-request | Stored datasets | Stored datasets are fetched | PASS |
| **TC06** | **GET** /datasets | Retrieve stored user data | Some other request | Error | Error: Improper request | PASS |
| **TC07** | User Selects Dataset | Selection | Data | Upload  Data | Dataset uploaded succesfully | PASS |
| **TC08** | User Selects Dataset | Selection | Data with imporoper featues | Upload  Data | Dataset uploaded succesfully | PASS |
| **TC09** | User Selects Features | Feature Engineering | Data | Selected Features | Proceed for ML | PASS |
| **TC09** | User Selects Features | Feature Engineering | Data with improper features | Improper Features | Do not proceed for ML | PASS |
| **TC10** | User users prediction APIs | Prediction | Row(s) of Data to be predicted | Prediction | Output of predicted values | PASS |
| **TC11** | User users prediction APIs | Prediction | Row(s) of improper Data to be predicted | Error | Improper: data passed | PASS |
| **TC12** | User users prediction APIs | Prediction | Row(s) of Data something lese | Error | Error | PASS |

## 9.3. Model Testing

To compare the two methods, the manual method and the project, we used a test problem indicative of a real-world use case for sentiment analysis: we used a text dataset with a categorical result of sentiment. The advantages of an accurate model are evident, as they allow the sentiment analysis to quickly identify cases of emotions in sentences.

### 9.3.1. Trying out this application

Started with this project, using their deep learning module to build a model automatically. Importing data and setting up a job was easy, although the tool has little flexibility on how to arrange and mark import data. Below is a screenshot of the results page that shows the results after < 15 minutes of training. AutoML has produced an impressive 81.3 per cent accuracy and 78.9 per cent recall on this task, without requiring a single line of code! It also provides an API to access the model we generated for use in the generation of predictions.

### 9.3.2. Using a Hand-Crafted Model with TensorFlow

Create our model from scratch using TensorFlow and TensorFlow Extended Library, which allows it easier to create and train models in TensorFlow. We used data increase on our training dataset by using vision transforms to construct a more reliable model, tuning the hyperparameters as well as downscaling and then upscaling the images in training iterations. With a few hours of work and ~15 minutes of training time, we obtained an accuracy of 82.7 per cent and a recall of 81.0 per cent, exceeding Google AutoML tests by 1–2 per cent.

### 9.3.3. Testing Results

While we have obtained superior results on this test problem through a handcrafted neural network over this process, it has also been significantly more time-consuming and complex. Even with the aid of TensorFlow Extended Library and TensorFlow, it took ~2 hours to set up and train our product. Once we were set up with AutoML, it took no more than 30 minutes to import our data and train the model without writing a single line of code. As far as usability and ease of use are concerned, AutoML wins hands down. However, when your dataset is not properly organized into named folders or you want to use custom validation sets, things get complicated. Today, AutoML is relatively linear in its interface and greatly reduces the flexibility and distinction that can be accomplished.

It fits well with "textbook" issues, but for many real-world problems with added complexity in inputs, names, or validation techniques, it just can't handle it today. Although we have achieved better precision modeling by hand, the significance of the 1–2 percent increase in accuracy we have achieved depends on the specific problem being resolved. In some cases, this may not be a significant difference, and in other cases, it may make or damage the value of the model. Overall, AutoML is a fine, easy-to-use tool to quickly build "good enough" models for standard classification tasks with no coding required. However, in its current form, it quickly breaks down problems with complicated data structure, custom labelling approaches or custom validation techniques often used in practice due to its applicability to real-world machine learning. It also serves as a black-box model, not revealing any of the internal intestines for inspection and subsequent adjustment as required for further training or development. In our experience, black-box models are almost always a bad idea when it comes to making predictions in the real world.

# 10. Conclusion

It's amazing to see the tremendous progress that has been made in automating deep learning over the last few years. It makes it more available to consumers and businesses; the potential of deep learning is made more accessible to the general public. But there's always room for improvement. Design search has become much more efficient; finding a single GPU network in a single training day is pretty amazing. However, our quest space is still very limited. Existing algorithms also use hand-designed constructs and building blocks, but tie them together differently! A successful and potentially ground-breaking future direction would be a far wider search for new architectures. These algorithms will reveal even more hidden deep learning secrets inside these large and complex networks. Of course, such a search space requires the efficient design of an algorithm. This new project is creating exciting challenges for the AI community and a chance for another advance in science. Overall, there are several options for using this project today. It just depends on whether you're going to play around with the algorithm you want and how much you're willing to pay to get some more code out of it. We live in an era where data growth outpaces our ability to make sense of it. This is explained not only by the current technological obstacles but also by our reliance on experts to carry out this mission. AutoML is an exciting field that has been on the spotlight and promises to alleviate this problem by intelligent automation of repetitive ML workflow tasks. Machine learning has become one of the main engines of the present era. The production line of machine learning models passes through different phases and stages, requiring a wide knowledge of several available tools and algorithms. However, as the amount of data generated daily continues to increase on an exponential scale, it has become important to automate this process. In this study, the state-of-the-art research effort in the field of AutoML frameworks has been comprehensively covered. We also highlighted research directions and open complexities that need to be addressed to achieve the vision and objectives of the AutoML project. We hope that our survey will serve as a useful resource for the community, both researchers and practitioners, to understand the challenges of the field and provide useful insights to further advance the state-of-the-art in several directions.

# 11. Future Enhancement

Essentially, the purpose of this project is to automate repetitive tasks such as pipeline development and hyperparameter tuning so that data scientists will focus more of their time on the business issue at hand. It also aims to make technology available to everyone rather than to a select few. This technology and data scientists can operate together to improve the ML cycle so that the true usefulness of machine learning can be used. Whether or not this project becomes a success depends mainly on its implementation and the progress made in this field. Nevertheless, the application is a big part of the future of machine learning. It will also handle most of the data-cleaning process It will greatly improve deep learning It will be applied to large data sets It will become human competitive It will change the nature of data science as we know this It is only a small part of a greater meta-learning revolution If you are employed in data science today or managing a team of data scientists, you will interpret and implement it as follows. Although more research efforts have been made in recent years to address the complexities of automated machine learning, there is still a range of open challenges and research directions that need to be tackled to achieve the ultimate goals and vision of the AutoML domain. In this segment, we highlight some of the issues that need to be tackled to improve the state of the art. Scalability: In reality, the main limitation of centralized systems for automating approaches to the CASH problem (e.g. Auto-Weka, Auto-Sklearn) is that they are closely linked to a machine learning library (e.g. Weka, sci-kit-learn, R) that can only work on a single node that makes them not applicable to large data volumes. In practice, as the amount of data generated daily continues to increase on an exponential scale, many distributed machine learning systems have recently been introduced.

Although some initial efforts have been made to develop a distributed automated framework for the CASH issue. However, the proposed distributed solutions are still simple and limited in their ability. More research efforts and new solutions are needed to address the challenge of automatically constructing and tuning machine learning models over massive datasets. Optimization Techniques: In practice, different AutoML frameworks use different hyperparameter optimization techniques for machine learning algorithms.

For example, Auto-Weka and Auto-Sklearn use the SMAC cross-validation technique during hyper-parameter configuration optimization and assessment. TPOT uses genetic programming and Pareto optimization to create candidate pipelines. It is difficult, in practice, to find a clear winner or one-size-fits-all technique. In other words, there is no single approach that can outperform all other techniques on different datasets with their different characteristics, types of search spaces and metrics (e.g. time and accuracy). There is, therefore, a crucial need to understand the pros and cons of these optimization techniques so that AutoML systems can automatically tune their hyperparameter optimization techniques or their strategy for exploring and traversing the search space. Such decision automating should provide improved performance over the selection process and depend on a defined strategy. Similarly, for the various meta-learning techniques adopted, there is no clear systematic method or evaluation metric to quantitatively evaluate and compare the effect of these techniques on the reduction of the search space.

## 11.1. User-friendliness

In general, most of the software and system currently in use cannot be called user-friendly. We still need advanced technical expertise to be applied and used. Such a challenge restricts its usefulness and wide acceptance among lay users and domain experts (e.g. doctors, accountants) who often have limited technical skills. One of the solutions to overcoming these problems may be the provision of an accessible and lightweight web interface for such a framework. Continuous delivery pipeline: continuous delivery is characterized as the development of a repeatable, consistent and incremental improvement mechanism for the transition of software from concept to the customer. Integrating machine learning models into continuous delivery pipelines for successful use has not gained much attention recently, as data scientists typically move them straight into the production environment with all the disadvantages that this method may have, such as no proper unit testing and integration testing. Data Validation: In this sense, most approaches in the literature concentrate on issue identification and user notification only. Nonetheless, an automated correction has not been studied in a good way, covering many potential domains of datasets and that the role of a data scientist in machine learning production.

Also, as possible data repair is an NP-Hard issue, there is a need to find more approximation techniques to solve this problem.

## 11.2. Data Preparation

There is a critical need to optimize the extraction process as it is known to be one of the most time-consuming sections of the pipeline. Throughout reality, most systems neglect the automation of moving data features to different domain spaces, such as performing main component analysis, or linear discriminant analysis, and improving model performance. Even, we assume that there is room for improvement of existing types of auto-encoders, such as Boltzmann Restricted Machines. Further research is needed, therefore, to try out different architectures and interpret them to be able to automate the choice of suitable encoders. Also, there are different methods for determining the value of a ranking, which is a very important part of the process of automating the selection process. However, there are no systematic comparative studies between these methods or effective recipes that can be recommended when using each of these techniques.

## 11.3. Model Deployment and Life Cycle

Recently, some tools and mechanisms have been put in place to make data scientist work easier and to automate machine learning development. In reality, however, there is still a need to incorporate these different systems along the entire pipeline. There is sufficient room for improvement, for example, about the automatic choice of good workflows appropriate to each problem and how to integrate more data comprehension, testing and planning techniques with workflows. For particular, these frameworks still do not provide the end-user with any smartness in the decision-making process that is the cornerstone of eliminating the role of a human being in the loop.

# Appendix A: Bibliography

# Appendix B: User Manual

1. Click on **Sign Up** and create an account for yourself. You can choose your appropriate **Email ID** and **Password.**
2. After the registration, you will be redirected to the **Main Page** of the application which is the **Data Set List Page**
3. Click on **Upload Dataset** and upload the **CSV** file to which you want to apply **Machine Learning** to.
4. Proceed next to Feature Engineering. Here, select the Input Variables that you want to predict from and Output Variables that need to be predicted
5. Wait for around 15 mins (depends on the dataset) and then you will see results along with the URL for predictions API
6. Send a POST request to the URL with your features for prediction. The URL will return a prediction JSON along with the parameters and related information.